Using the primal-dual interior point algorithm within the branch-price-and-cut method

Munari, P & Gondzio, J 2013, 'Using the primal-dual interior point algorithm within the branch-price-and-cut method' Computers and Operations Research, vol 40, no. 8, pp. 2026-2036.

Using the primal-dual interior point algorithm within the branch-price-and-cut method

Pedro Munari[a,1], Jacek Gondzio[b]

[a] *Instituto de Ciências Matemáticas e de Computação, University of São Paulo, Av. Trabalhador São-carlense, 400 - Centro, Cx. Postal 668, CEP 13560-970, São Carlos-SP, Brazil*
[b] *School of Mathematics, The University of Edinburgh, James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom*

Abstract

Branch-price-and-cut has proven to be a powerful method for solving integer programming problems. It combines decomposition techniques with the generation of both columns and valid inequalities and relies on strong bounds to guide the search in the branch-and-bound tree. In this paper, we present how to improve the performance of a branch-price-and-cut method by using the primal-dual interior point algorithm. We discuss in detail how to deal with the challenges of using the interior point algorithm with the core components of the branch-price-and-cut method. The effort to overcome the difficulties pays off in a number of advantageous features offered by the new approach. We present the computational results of solving well-known instances of the Vehicle Routing Problem with Time Windows, a challenging integer programming problem. The results indicate that the proposed approach delivers the best overall performance when compared with a similar branch-price-and-cut method which is based on the simplex algorithm.

Keywords: branch-price-and-cut, column generation, primal-dual interior point algorithm, vehicle routing problem

1. Introduction

The branch-price-and-cut method has been widely used for solving integer programming models in which a special structure can be identified in the coefficient matrix. This structure is exploited by a reformulation technique, e.g. the Dantzig-Wolfe decomposition, that usually leads to a formulation with a stronger linear relaxation when compared with the linear relaxation of the original model. By using this stronger formulation within a branch-and-bound tree, we obtain the branch-and-price method. Since the reformulation may have a huge number of variables, a column generation algorithm is used to solve the linear relaxation. For this reason, the branch-and-price method is also known as integer programming column generation. In some cases, valid inequalities should also be added to the reformulated model to get even better bounds from the linear relaxations with the aim of improving the branch-and-bound search, which leads to the branch-price-and-cut method. See [1–3] for comprehensive surveys on these methods.

In the vast majority of branch-price-and-cut implementations presented in the literature, a simplex-type method is used to solve the linear programming problems at each node. Hence, the generation of columns and valid inequalities

Email addresses: munari@icmc.usp.br (Pedro Munari), J.Gondzio@ed.ac.uk (Jacek Gondzio)
[1] Supported by FAPESP (São Paulo Research Foundation, Brazil) under the projects 2008/09040-7 and 2012/05486-6.

are typically based on optimal solutions of these problems. Particularly in the column generation case, the use of optimal dual solutions which are extreme points of the dual feasible set may adversely affect the performance of the method. The reason is the high oscillation between extreme points from one iteration to another, which may result in slow convergence as well as a temporary stalling of the algorithm, specially in highly degenerate formulations. To overcome this behavior, different strategies have been proposed in the literature which use non-extremal dual solutions so that more stable column generation procedures are obtained [4–8]. Other strategies, such as Dynamic Constraint Aggregation, have also been used with the same purpose [9]. Using non-extremal primal solutions to generate valid inequalities has shown to be more effective as well, since deeper cuts are obtained and a smaller number of them are usually needed [10–13]. In this paper, we investigate the use of the primal-dual interior point method (PDIPM) to provide non-optimal solutions which are well-centered in the feasible set. The computational experience provides evidence that these solutions are beneficial to the generation of columns and valid inequalities.

Although very successful in many other fields related to linear programming, interior point methods do not seem to have made a big impact in the integer programming context. It is probably because the standard integer programming methodologies were originally proposed at time when the simplex method was the only efficient algorithm available for solving linear programming problems and, hence, they were biased to the features available in this method. Moreover, until a few years ago, interior point methods were not able to reoptimize a problem after carrying out modifications to the data as efficiently as a simplex type method, a scenario that has changed in the last years with the development of efficient warmstarting techniques for interior point methods [14–16]. Another reason may be due to previous unsuccessful attempts of straightforwardly replacing a simplex type method by an interior point method. These two methods are very different [17] and, hence, should not be used in the same way.

In this paper, we address the several facets of using the PDIPM within a branch-price-and-cut method. We discuss in detail how to modify the core components of this method in order to exploit the advantageous features offered by the interior point algorithm. We believe it is an appropriate time for an investigation like this, as interior point methods have achieved a mature status concerning both theory and computational implementations. To verify the proposed approach, we present computational results for well-known instances of the vehicle routing problem with time windows (VRPTW). It is a classical application of integer programming that is widely used for testing new algorithms due to its difficulty. It is worth mentioning that the issues related to the integration of the interior point algorithm with the branch-price-and-cut method which we discuss in this paper are not limited to or specialized for the VRPTW and can be straightforwardly used in other combinatorial optimization applications.

The remainder of this paper is organized as follows. In Section 2, we briefly describe the fundamental concepts of the PDIPM and present a literature review of previous attempts of combining interior point algorithms with integer programming methodologies. In Section 3, we address all the issues involved in the use of the PDIPM in the branch-price-and-cut method and propose how to deal with them. The VRPTW is briefly described in Section 4 and the results of computational experiments with the new approach are presented in Section 5. The conclusion and potential further developments are presented in Section 6.

2. Interior point methods and integer programming

Starting with Karmarkar's projective algorithm [18], interior point methods have quickly and strongly evolved in the last few decades [17, 19]. They have been successfully applied not only to solving linear programming problems but also in many other areas such as quadratic, semi-definite, and conic programming. However, in spite of the close relationship between linear programming and integer programming, interior point methods have not showed a similar impact on integer programming. In this section we briefly describe the fundamental concepts of the PDIPM and then present a literature review of different attempts to use interior point methods within integer programming approaches that rely on linear programming relaxations.

2.1. The primal-dual interior point method (PDIPM)

In this paper, we consider a linear programming problem represented by the following primal (P) and dual (D) formulations

$$\begin{array}{llll} (P) & \min & c^T\lambda & \\ & \text{s.t.} & A\lambda = b & \\ & & \lambda \geq 0, & \end{array} \qquad \begin{array}{llll} (D) & \max & b^T u & \\ & \text{s.t.} & A^T u + s = c & \\ & & s \geq 0, & \end{array} \tag{2.1}$$

where A is a full row rank matrix of columns $a_j \in \mathbb{R}^m$, $j \in N = \{1, \ldots, n\}$, $\lambda \in \mathbb{R}^n$ is the vector of primal variables, $u \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$ are the vectors of dual variables, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are parameters of the problem, and $0 < m \leq n$. The first order optimality conditions associated to this pair of problems are given by

$$b - A\lambda = 0 \tag{2.2}$$

$$c - A^T u - s = 0 \tag{2.3}$$

$$\Lambda S e = 0 \tag{2.4}$$

$$(\lambda, s) \geq 0, \tag{2.5}$$

where $\Lambda = diag(\lambda_1, ..., \lambda_n)$, $S = diag(s_1, ..., s_n)$, and $e = (1, 1, ..., 1)$. The PDIPM is based on the perturbation of the optimality conditions, in which equation (2.4) is replaced by $\Lambda S e = \mu e$ where $\mu > 0$ is the *barrier parameter*, or *duality measure*. This parameter is gradually driven to zero throughout the iterations so that a primal-dual pair of optimal solutions satisfying (2.2)-(2.5) is obtained at the end of the algorithm. For a given value of μ, the perturbed optimality conditions have a unique solution, which is called a μ-center. The set composed of all μ-centers is called a central-path. Instead of strictly satisfying the perturbed optimality conditions, the iterates of the PDIPM belong to a neighborhood of the central path. The idea of the neighborhood is to keep the iterates well-centered and in a safe area so that all variables approach their optimal values with a uniform pace. Different neighborhoods have been proposed in the literature. For instance, the symmetric neighborhood requires all the iterates to satisfy (2.2) and (2.3) and further belong to the set

$$\{(\lambda, u, s) \mid \gamma\mu \leq \lambda_j s_j \leq (1/\gamma)\mu, \forall\, j = 1, \ldots, n\},$$

for a given parameter $\gamma \in (0, 1)$.

In a given iteration of the PDIPM, the corresponding iterate (λ, u, s) is modified by using the directions $(\Delta\lambda, \Delta u, \Delta s)$ which are obtained by solving the following Newton step equations

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta u \\ \Delta s \end{bmatrix} = \begin{bmatrix} \xi_\lambda \\ \xi_s \\ \sigma\mu e - \Lambda S e \end{bmatrix}, \tag{2.6}$$

where σ is a parameter used to reduce the complementarity gap $\lambda^T s$ of the next iterate, and vectors $\xi_\lambda = b - A\lambda$ and $\xi_s = c - A^T u - s$ are the primal and dual infeasibilities of the current iterate, respectively. After obtaining the directions, the step-sizes α_P and α_D are computed for the primal and dual components, respectively, in order to guarantee that the next iterate will belong to the neighborhood of the central path. The next iterate is then given by $(\lambda + \alpha_P\Delta\lambda, u + \alpha_D\Delta u, s + \alpha_D\Delta s)$. For a full description of the method and a discussion about its main theoretical and implementation issues, the reader is referred to [17].

2.2. Interior point methods within integer programming methodologies

Few attempts have been presented in the literature regarding integer programming methodologies that are based on interior point algorithms. The first implementations in this sense started only in the beginning of the nineties, with the pioneering works by Mitchell and his collaborators. In [10], the primal projective interior point algorithm is combined with a cutting plane method and employed to solve the perfect matching problem. The authors propose to use early termination when solving the linear programming problems and present how to deal with several issues such as how to obtain a new iterate after adding constraints and columns to a problem that has just been solved. As shown in the computational results, the proposed approach resulted in a reduction of the number of iterations and the number of calls to the separation subproblem, although the CPU times were not competitive with a cutting plane method based on the simplex method. The authors associate this behavior to the difficulty in calculating the projections required by the projective interior point method used in their experiments, as well as to the inefficient warm-starting strategy used after the addition of cutting planes. Indeed, a more efficient interior point cutting plane method was obtained a few years later, by using the PDIPM and improved warm-starting strategies [11, 12]. According to the computational results presented for two integer programming applications, namely the linear ordering problem and the max-cut problem, the interior point approach was competitive with a cutting plane algorithm based on the simplex method.

The use of an interior point algorithm within a branch-and-bound method was first exploited by Borchers and Mitchell [20]. The authors observed that the PDIPM tends to quickly find feasible solutions with good objective values, but then may spend considerable time approaching a solution accurate enough to meet the termination criteria. In addition, the iterates tend to converge steadily to an optimal solution and, hence, the authors propose to use an early branching strategy, which consist in not solving the node to optimality, but stopping as soon as it becomes apparent that the optimal solution of the node includes an integer variable at a fractional value. The authors present preliminary computational results on a set of mixed-integer programming problems. The interior point branch-and-bound was competitive in about half of the instances in comparison with a state-of-the-art branch-and-bound algorithm based on the simplex method. In the remaining instances, the latter dominates as the simplex method was significantly faster than the PDIPM in solving the LP relaxations of these problems. The authors associate the inferior performance of the

new approach to the lack of efficient warm-start techniques for the PDIPM, a situation that has changed significantly since then.

In [21], du Merle et al. propose the use of the analytic center cutting plane method (ACCPM) within the branch-and-price method for solving the minimum sum-of-squares nonhierarchical clustering problem, a constrained hyperbolic program in 0-1 variables (see [22] for a recent investigation of this problem). In the ACCPM approach, an interior point method is used to obtain approximate analytic centers of the localization set, which corresponds to the dual feasible set with an additional inequality for the best bound found so far (see [6] for further details of the method). According to the computational results for several fairly large data sets publicly available, the proposed combination is very promising and allows the exact resolution of substantially larger instances than those treated before. Recently, a similar combination was investigated in [13] for integer programming problems. The authors show how to modify the ACCPM so that it can warm-start after branching as well as after the addition of columns in the primal problem. In the computational results presented in the paper, the authors have used randomly generated instances of the bin packing problem and the capacitated facility location problem. The results show that for the bin packing instances, the branch-and-price method using the ACCPM results in a number of nodes that is comparable with the branch-and-price method using the standard column generation. However, the former requires on average 58% of the CPU time required by the latter. When comparing the two methods for the facility location instances, the method using ACCPM explores less nodes and requires less computational time. Moreover, it is consistently better than the method using the standard column generation in making fewer calls to the pricing subproblems and requiring less computational time per node.

3. Main issues of an interior point branch-price-and-cut method

In this section, we discuss the issues of using the PDIPM within the branch-price-and-cut method. We address the main elements involved in this combination, namely the column generation technique, the separation and addition of valid inequalities, and branching. As it will be seen in the computational results, the effort to overcome the challenges pays off in a number of advantageous features offered by the new approach.

3.1. Primal-dual column generation

The column generation technique is an iterative method applied to solve a linear programming problem in which the coefficient matrix has a huge number of columns, but these columns can be generated by a rule that is known in advance. The idea is to keep only a subset of columns in the coefficient matrix, and reach an optimal solution without explicitly generating all the columns. In the column generation literature, the linear programming problem we are interested in is called the *(continuous) master problem* (MP). Typically it has a finite but possibly an exponential number of variables. Consider an MP represented as the primal problem (P) in (2.1). The auxiliary problem containing only a subset of variables of the MP is called the *restricted master problem* (RMP), and is represented here by

$$z_{RMP} := \min \quad \sum_{j \in \overline{N}} c_j \lambda_j, \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{j \in \overline{N}} a_j \lambda_j = b, \tag{3.1b}$$

$$\lambda_j \geq 0, \qquad \forall j \in \overline{N}, \tag{3.1c}$$

where $\overline{N} \subset N$ and the remaining elements are the same as described in Section 2.1. In this context, λ_j is known as a master variable. The rule used to generate a column is given by the *oracle*, also known as the *pricing subproblem*, represented by

$$z_{SP}(\overline{u}) := \min\{0; c_j - \overline{u}^T a_j \mid j \in N\}, \tag{3.2}$$

where $\bar{u} \in \mathbb{R}^m$ is a dual solution associated to constraints (3.1b), and c_j and a_j are the variables of the subproblem. By using the dual solution $\bar{u}$, the oracle must be able to generate a column of the MP that is not in the RMP, and this new column should have a negative reduced cost. In case it is not possible to generate such column, the optimal solution of the RMP is also optimal for the MP and the column generation procedure terminates.

In the standard column generation, the RMP is solved to optimality in every outer iteration, so that the oracle receives an extreme point of the dual set. However, the use of extreme points causes instability in the column generation. Indeed, the high oscillation between extreme points of consecutive outer iterations typically results in slow progress of the algorithm, specially during the last iterations. Different techniques have been proposed in the literature in order to obtain non-extremal dual points which lead to more stable strategies [4–9, 23]. Here, we use the *primal-dual column generation technique*, which relies on the PDIPM to solve the RMP, so that non-optimal, well-centered dual points are obtained, leading to a naturally stable strategy. Recently, promising computational results were presented in [8] for this technique when solving master problems obtained from linear-relaxed reformulations of three classical integer programming problems: the cutting stock problem, the vehicle routing problem with time windows, and the capacitated lot sizing problem with setup times. Since the main goal of the investigations presented in [8] was to check the performance of the column generation algorithm, the authors did not consider obtaining integer solutions in their computational experiments. In this paper we extend the primal-dual column generation method and use it within the branch-price-and-cut approach which is able to solve the integer programming problems to optimality. In the remainder of this section, we summarize the main aspects of the primal-dual column generation technique, following the discussion presented in [8].

In a given iteration of the primal-dual column generation technique, the first step is to obtain a suboptimal solution of the current RMP, *i.e.*, a primal-dual feasible solution $(\tilde{\lambda}, \tilde{u})$ which is not optimal but must satisfy $0 < (c^T\tilde{\lambda} - b^T\tilde{u}) \leq \varepsilon(1 + |c^T\tilde{\lambda}|)$, for some tolerance $\varepsilon > 0$ that determines the distance of the suboptimal solution to optimality. For this reason, a suboptimal solution is also known as ε-optimal solution. We additionally require the solution to be well-centered in the feasible set. This comes at no extra effort since the PDIPM keeps the iterates inside a neighborhood of the central path (see Section 2.1).

The tolerance ε is dynamically adjusted in function of the relative gap in the current iteration of the column generation procedure. First, we compute the relative gap

$$gap = \frac{c^T\tilde{\lambda} - (\kappa z_{SP}(\tilde{u}) + b^T\tilde{u})}{1 + |c^T\tilde{\lambda}|},$$

where $\kappa > 0$ is a constant that must satisfy $\kappa \geq \sum_{j \in N} \lambda_j^*$, for an optimal solution λ^* of the master problem (this value is usually known in the context of a reformulation). Having computed the relative gap, ε is set as $\min\{\varepsilon_{\max},\ gap/D\}$,

where the scalar $\varepsilon_{\max} > 0$ imposes an upper bound for the tolerance value (*e.g.*, $\varepsilon_{\max} = 1.0$) and $D > 1.0$ is the degree of optimality that controls the reduction in ε in function of the relative gap. Notice that it is an intuitive way of dynamically adjusting the distance to optimality if we observe that during the first iterations of the column generation procedure (in which the gap is relatively large) the corresponding RMPs are merely rough approximations of the MP and, hence, there is no reason for solving them to optimality. On the other hand, as the gap gets closer to zero, a better approximation of the MP is available in the RMP and, hence, it becomes advantageous to get a point that is closer to optimality.

The primal-dual column generation procedure is summarized in Algorithm 1. It is very similar to the algorithm of the standard column generation, except for lines 4 and 9 in which we require a suboptimal and well-centered solution, and set the relative distance to optimality ε, respectively. In line 6, $ORACLE(\tilde{u})$ denotes the oracle procedure in charge of generating one or more columns with a negative reduced cost (if any).

Algorithm 1: Primal-Dual Column Generation Method

1. **Input:** Initial RMP; parameters κ, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
2. set LB $= -\infty$, UB $= \infty$, $gap = \infty$, $\varepsilon = 0.5$;
3. **while** $(gap \geq \delta)$ **do**
4. find a well-centered ε-optimal solution $(\tilde{\lambda}, \tilde{u})$ of the RMP;
5. UB $= \min(\text{UB}, c^T\tilde{\lambda})$;
6. call $ORACLE(\tilde{u})$;
7. LB $= \max(\text{LB}, \kappa z_{SP}(\tilde{u}) + b^T\tilde{u})$;
8. $gap = (\text{UB} - \text{LB})/(1 + |\text{UB}|)$;
9. $\varepsilon = \min\{\varepsilon_{\max},\ gap/D\}$;
10. **if** $(z_{SP}(\tilde{u}) < 0)$ **then** add the new columns to the RMP;
11. end(while).

A proof of convergence for this algorithm is given in [8]. Furthermore, according to the computational results reported in the same paper, the primal-dual column generation technique outperforms the standard column generation as well as the analytic center cutting plane method, considering the number of iterations and CPU time, on average, for all the applications used in the experiments. Moreover, the results show that the larger the instances, the better is the relative performance of the primal-dual approach. In summary, the authors conclude that by exploiting the advantages offered by the interior point algorithm, an improved column generation procedure was obtained. With this in mind, in the next section we investigate the use of the PDIPM in the generation of valid inequalities, in particular for the branch-price-and-cut method.

3.2. Primal-dual column and cut generation

For some applications, valid inequalities can be used to improve the bounds provided by the master problems and consequently reduce the number of nodes in the search tree. Different types of valid inequalities are available in the literature and their effectiveness usually depends on the problem. Besides, in the context of column generation, the addition of valid inequalities is not a trivial task as it may drastically increase the computational cost of solving the pricing subproblem and, hence, a good trade-off must be achieved [1, 2].

Let (3.1) represent the RMP in a given iteration of the column generation procedure (which may be the last one). Assume that the current primal solution is fractional so that a *separation procedure* can be used to generate a subset

of violated valid inequalities. By adding these valid inequalities to the problem, we obtain

$$z'_{RMP} := \min \quad \sum_{j \in \overline{N}} c_j \lambda_j, \tag{3.3a}$$

$$\text{s.t.} \quad \sum_{j \in \overline{N}} a_j \lambda_j = b, \tag{3.3b}$$

$$\sum_{j \in \overline{N}} h_j \lambda_j \leq d, \tag{3.3c}$$

$$\lambda_j \geq 0, \qquad \forall j \in \overline{N}, \tag{3.3d}$$

where $d \in \mathbb{R}^{m'}$, $h_j \in \mathbb{R}^{m'}$, for all $j \in \overline{N}$, and $m' > 0$ is the number of valid inequalities. Let $\bar{\sigma} \in \mathbb{R}^{m'}$ be a dual solution associated to (3.3c). This dual solution must be taken into account in the pricing subproblem. Depending on the type of the valid inequalities, this requirement may result in simple changes of the subproblem, which affect only its objective function, *i.e.*,

$$z'_{SP}(\overline{u}, \overline{\sigma}) := \min\{0; c_j - \overline{u}^T a_j - \overline{\sigma}^T h_j \mid j \in N\}. \tag{3.4}$$

However, some valid inequalities may lead to more complex changes. For example, when the coefficients h_j are dependent on certain features of the solution which is used to generate the column a_j, the objective function of the subproblem may become non-linear. In some cases, the function can be linearized and the subproblem can be expressed as in (3.4) but it results in the addition of new variables and constraints that may increase the complexity of the subproblem. A very important concern is to keep the number of inequalities small in order to minimize the impact of the changes in the subproblem. Due to these complications, this second type of valid inequalities has rarely been studied in the literature [2, 24]. However, its use is justified in cases that it leads to stronger valid inequalities. In Section 4.2, a valid inequality of this second type will be used in the vehicle routing problem with time windows, following [25].

Separation procedures are usually called at the optimal solution of the master problem, *i.e.*, after the column generation procedure has finished. However, as it was already mentioned in Section 2.2, calling the separation procedure before reaching optimality is likely to result in deeper cuts, as well-centered points in the interior of the feasible set are used to generate the valid inequalities. Besides, this strategy facilitates the warm-start when an interior point method is used to solve the RMP, as discussed in Section 3.4. With these observations in mind, we propose to modify the oracle of the primal-dual column generation algorithm (Algorithm 1) so that two operations are available. Either new columns are generated by calling the pricing subproblem, or new constraints are generated by calling the separation subproblem. The separation subproblem is in charge of generating valid inequalities for the RMP, based on the current primal (feasible) solution. A few drawbacks are associated to this early search for valid inequalities: (*i*) it may be too early to call the separation subproblem and it will only waste time without returning any violated inequality; (*ii*) the search may find too many valid inequalities, as an inequality may be violated by the current iterate but not by an optimal solution. As observed in [12], these disadvantages may be minimized by keeping the iterates well-centered in the feasible set, and by using a tolerance threshold to start generating valid inequalities, as discussed below. Furthermore, the gain of using the early search strategy is likely to overcome these potential drawbacks.

Algorithm 2 is an extension of Algorithm 1 which takes the generation of valid inequalities into account. The oracle procedure was modified so that the primal solution of the RMP is also sent as a parameter to the oracle in line 7. This procedure is not detailed in the algorithm, as different strategies may be used to define how the pricing and the separation subproblems should be called. For instance, valid inequalities should be generated only after the relative gap falls below a tolerance threshold ε_c. In other words, the oracle should call the separation subproblem only if $gap < \varepsilon_c$. In practice, $\varepsilon_c = 0.1$ or $\varepsilon_c = 0.01$ are typically good choices. The value of ε_c may also be dynamically adjusted according to the maximum violation and the number of violated constraints, as suggested in [11]. In addition, it is important to avoid calling the separation subproblem in two consecutive outer iterations. By alternating between the two types of subproblems, more accurate points are sent to the oracle and a better overall performance is likely to be achieved in practice.

Algorithm 2: Primal-Dual Column and Cut Generation Method

1. **Input:** Initial RMP; parameters κ, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
2. set LB $= -\infty$, UB $= \infty$, $gap = \infty$, $\varepsilon = 0.5$;
3. **while** ($gap \geq \delta$) **do**
4. find a well-centered ε-optimal solution $(\tilde{\lambda}, \tilde{u})$ of the RMP;
5. **if** in the last iteration new cuts were added to the RMP **then** UB $= c^T\tilde{\lambda}$;
6. **else** UB $= \min(\text{UB}, c^T\tilde{\lambda})$;
7. call $ORACLE(\tilde{\lambda}, \tilde{u}, \tilde{\sigma})$;
8. **if** new cuts were generated **then** add them to the RMP;
9. **else** LB $= \max(\text{LB}, \kappa z'_{SP}(\tilde{u}, \tilde{\sigma}) + b^T\tilde{u})$;
10. $gap = (\text{UB} - \text{LB})/(1 + |\text{UB}|)$;
11. $\varepsilon = \min\{\varepsilon_{\max},\ gap/D\}$;
12. **if** ($z'_{SP}(\tilde{u}, \tilde{\sigma}) < 0$) **then** add the new columns to the RMP;
13. end(while).

Although the early search for valid inequalities has been exploited in cutting plane methods in the literature, the reader should notice that in the context addressed here, the separation procedure will be called in the course of the column generation algorithm. It is an interesting situation if we notice that we will cut-off part of the primal feasible set, which may be expanded again by generating new columns in the next call to the pricing subproblem. Moreover, if the valid inequalities are of the second type discussed at the beginning of this section, the subproblem may quickly become more and more difficult to solve as m' increases. Hence by reducing the number of valid inequalities due to the use of non-optimal, well centered solutions, we hope to improve the overall performance of the algorithm.

3.3. Branching

Branching is another core element of the branch-price-and-cut method. Given a fractional solution of the linear relaxation associated to a node, the branching procedure usually splits the node into two child nodes, each one having a subset of the feasible set. Different branching rules are available and their efficiency is typically problem-dependent. Furthermore, the standard branch-and-bound strategy of selecting only one fractional component to branch is usually inefficient and can be prohibitive in a branch-price-and-cut context, so more elaborate strategies must be used. On the other hand, it is essential to use a branching rule that is compatible with the column generation scheme, *i.e.*, a rule that does not increase too much the difficulty of solving the pricing subproblem [26]. For instance, the branching rule that is typically used for the vehicle routing problem with time windows is imposed on the pricing subproblem, without

compromising the performance of solving the subproblem [27]. The initial RMP of a (child) node, corresponds to the last RMP of its parent node without the columns that violate the new constraints imposed by the branching rule.

When solving the linear relaxation of a node by an interior point method, *early branching* is likely to result in a better overall performance of the branch-and-bound search [20]. Similarly to what is done in the column and cut generation, the early branching consists in calling the branching procedure before reaching the optimality of the linear relaxation (master problem). It is based on the fact that, when solving a standard linear programming problem, an interior point method gets close to optimality quickly but then may need a considerable effort to attain the required accuracy of the optimal solution. Indeed many components of the solution vector may reach the values which are very close to the optimum after merely a few interior point iterations. As showed in [20], it can be useful in reducing the CPU time of the overall method. However, it is not clear whether the use of early branching in combination with column and cut generation is beneficial. In fact, the columns and cuts added in the very last iterations may be decisive for the values of the components of an optimal solution and, hence, the early branching may be misguided by wrong estimates of the solution values.

Here we propose to deal with the branching operation in two steps. In the first step, called pre-processing, we aim at quickly obtaining a non-optimal solution of the master problem of the current node so that the first evidence of a potentially good branching decision is taken from it. If the node is not eligible for pruning, then the branch is carried out and the second step is not necessary. Otherwise, the second step is started in which the master problem is solved to optimality. Different strategies may be used to solve the problem quickly, such as setting a loose optimality tolerance ε_b (where $\delta < \varepsilon_b < \varepsilon_c$) in the column and cut generation procedure, and using a heuristic method to solve the pricing subproblem as it is usually the most time-demanding procedure.

3.4. Warmstarting strategy

The core elements of a branch-price-and-cut method discussed in the previous sections share a common feature: they typically involve the solution of a linear programming problem that is a simple modification of another linear programming problem that has been already solved, *i.e.*, these problems are *closely-related*. For instance, after solving an RMP in a given iteration of the column and cut generation procedure, the RMP to be solved in the next iteration differs from that of the previous one by having either new columns or new constraints in the primal formulation. Any information available from the solution of the previous RMP may be useful to speed up the solution of the new one. A technique that is able to properly exploit such information and obtain an initial solution of the problem with beneficial characteristics in relation to a default initial solution is called a *warmstarting technique*.

The first attempts of warmstarting interior point methods were biased to the strategy used in simplex-type methods. Namely, after solving a problem to optimality the optimal solution is used as a warmstart for solving the modified problem. In the context of simplex-type methods such approach typically works well, as these methods consider only points that are vertices of the feasible set and hence a few iterations are typically enough to reoptimize the problem after a mild modification of it. For interior point methods we have the opposite situation, vertices of the feasible set must be avoided. In fact, starting from an optimal solution of a closely-related problem is likely to cause a worse performance of the method in comparison with starting from a default initial point. As already observed in the literature [14], this happens because the optimal solution is typically a badly-centered point that is very close to

the boundary of the feasible set. Hence, the algorithm will spend a relatively large number of iterations to regain centrality, a fundamental feature of an interior point method. Besides, numerical instabilities may affect the solution of the linear system of equations due to very small components that are usual in optimal solutions. By observing these drawbacks, different strategies have been presented for interior point methods. Many of them store one or more iterates of the closely-related problem that has been previously solved [11, 14, 28–31]. The stored iterates are then adjusted in order to become well-centered and sufficiently feasible. There are also warmstarting strategies that work in a different manner, in which the problem is modified by the addition of slack variables and penalization factors so that the optimal solution of the closely-related problem can be used as a warmstart [15, 16].

In the interior point branch-price-and-cut method presented in this paper we follow the ideas proposed in [14, 29]. RMPs are solved approximately with loose accuracy requirements. Non-optimal and well-centered solutions are thus obtained and they provide high-quality warmstarting points when new problems in the primal-dual column and cut generation process are created. The solution $(\tilde{\lambda}, \tilde{u})$ sent to the oracle (see Section 3.2) is the same as the one stored for constructing a warmstarting point to be used in the next outer iteration, except when the relative gap becomes less than a threshold value ε_{ws} (*e.g.*, $\varepsilon_{ws} = 10^{-3}$) which means the solution is too close to the boundary for leading to a good warmstart. In this particular case, we store the iterate when for the first time the relative gap falls below ε_{ws} and continue solving the problem until the predefined optimality tolerance is reached. Before solving the next RMP, the stored point is modified to reduce the primal and dual infeasibilities, and multiple centrality correctors [32, 33] are applied to improve the centrality of the warmstarting point.

4. The vehicle routing problem with time windows (VRPTW)

The VRPTW is a widely studied integer programming problem and covers a broad range of real-world applications. All the time, many companies over the world are looking for the best routes for delivering/collecting products to/from customers, which are usually spread in a certain neighborhood, city or even larger regions. The VRPTW can be used to model these situations (see [34–37] for interesting examples) and the efficient solution of real-life problems is crucial for many businesses. Moreover, the VRPTW is important as a benchmarking for testing new solution strategies, because it is considered a very difficult integer programming problem.

In the standard VRPTW, a set of vehicles is available in a single depot and these vehicles must be used to visit a set of customers $C = \{1, \ldots, m\}$ in order to satisfy their respective demands. The problem consists in determining a set of minimum cost routes for the vehicles, satisfying the following requirements. Each customer must be visited exactly once and the arrival of the vehicle must satisfy a time window $[w_i^a, w_i^b]$, *i.e.*, the vehicle cannot arrive after time w_i^b to service a customer, and it should wait until time w_i^a in case it arrives too early. The number of customers a vehicle can serve is limited by its maximum capacity, and the vehicle must return to the depot after visiting the customers covered by its route. We assume the vehicles are identical and they are available in a sufficient number to service all the customers. The cost of a route is given by the total distance traveled by the vehicle. For a detailed description of the problem see e.g. [38, 39].

Although a compact formulation of the VRPTW has been proposed in the literature (see [38]), its linear relaxation provides a poor bound for the value of an optimal integer solution. As a consequence, a pure branch-and-bound

strategy or even a branch-and-cut method are usually inefficient in practice. A better bound can be obtained by recurring to the linear relaxation of an extended formulation of the VRPTW, given by the following integer master problem with a set-partitioning structure

$$\min \quad \sum_{j \in N} c_j \lambda_j \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{j \in N} a_j \lambda_j = 1, \tag{4.1b}$$

$$\lambda_j \in \{0, 1\}, \qquad \forall j \in N, \tag{4.1c}$$

where $a_j = (a_{1j}, \ldots, a_{mj})^T$ is a column constructed from a feasible route, c_j is the corresponding cost to visit each customer in the route and then come back to the depot, and N is the set of indices of all feasible routes for the problem. The coefficients in column a_j are given by $a_{ij} = 1$ if route j visits customer i and $a_{ij} = 0$ otherwise.

Formulation (4.1) is typically solved by a branch-and-price strategy. At each node, the integrality of the master variable λ is dropped and the resulting linear programming relaxation is solved by the column generation technique. To generate columns, feasible routes are obtained by solving an elementary shortest path problem with resource constraints (ESPPRC). Valid inequalities are also generated in this context, in order to improve the bounds that are provided by the master problem. In the remainder of this section, we briefly address the main features of our implementation of the branch-price-and-cut method for solving the VRPTW.

4.1. Solving the ESPPRC

The most successful implementations for solving the ESPPRC are based on a dynamic programming algorithm. Specifically, it is a label-setting algorithm that was proposed in [40, 41] and has been significantly improved in the last years by the development of different techniques, such as the identification of unreachable nodes [42] and the bidirectional label extension with resource bounding [43]. Also, heuristics and meta-heuristics have been used to speed up the implementations [27, 44]. We have followed all these references in our implementation of the label-setting algorithm for solving the ESPPRC. In addition, the algorithm is initialized by the set of routes associated to the columns in the restricted master problem. Up to K_1 columns are used and we select those associated to the smallest values s_j (the slack dual component related to λ_j), for all $j \in \overline{N}$. We also use an adaptation of the Clarke and Wright heuristic [34, 45] to construct initial routes that are inserted into this set. Furthermore, three improvement heuristics are applied upon the best paths in the set. The first one consists in, given a route, removing one customer at a time from it. In the second heuristic, we try to insert one customer that is not in the route, between each two customers in the route. The third heuristic is given by swapping any two consecutive customers of a route. In all methods, we try to improve up to K_2 routes of the initial set, those with the least reduced costs. Each heuristic is called once and a modified route is inserted into the set if it has a negative reduced cost.

As proved in [46], the ESPPRC is a strongly NP-hard problem. Finding the optimal route at each call to the oracle is computationally expensive and should be avoided in the column generation procedure, as the optimal solution is needed only at the last outer iteration. As proposed in [27], we apply a few simplifications to the exact label-setting algorithm, so that it becomes a relatively quick heuristic. One simplification is given by imposing a limit K_d on the number of vertices for which we verify the dominance criteria that is used to eliminate paths. Besides, a given label is

extended to at most K_e adjacent vertices. Another simplification consists in setting a limit K_l for the total number of labels generated by extensions. The values of these three parameters are dynamically chosen according to the relative gap in the column generation procedure. When the relative gap falls below the column generation optimality tolerance δ, we discard these simplifications and call the exact label-setting algorithm.

4.2. Valid inequalities for the VRPTW

Different types of valid inequalities are available for the VRPTW. Some of them are based on the variables of the compact formulation of the problem [27, 47, 48], which do not increase the difficulty of solving the pricing subproblem in a branch-and-price approach. Others are based on the extended formulation [24, 25], which leads to tighter bounds at the cost of having a more difficult pricing subproblem. In this paper, we use a valid inequality of this second type, namely the *subset row* (SR) inequality proposed by Jepsen et al. [25]. Consider an RMP associated to problem (4.1) and recall that C is the set of customers in the problem. Given a subset $S \subset C$ and a scalar θ such that $0 < \theta \leq |S|$, the corresponding SR inequality is defined as

$$\sum_{j \in \overline{N}} \Big\lfloor \frac{1}{\theta} \sum_{i \in S} a_{ij} \Big\rfloor \lambda_j \leq \Big\lfloor \frac{|S|}{\theta} \Big\rfloor, \tag{4.2}$$

where $\overline{N} \subset N$ is the set of indices of master variables in the RMP. In [25], the authors suggest the choices $|S| = 3$ and $\theta = 2$, as SR inequalities using other values seldom appeared in the computational experiments carried out on benchmarking instances. In this particular case, the SR inequality for a given subset S is given by

$$\sum_{j \in \overline{N}_S} \lambda_j \leq 1,$$

where $\overline{N}_S \subset \overline{N}$ is the subset of routes which visit at least $\theta = 2$ customers that belongs to S.

The addition of the SR inequalities to the RMP can significantly increase the difficulty of solving the subproblem, in particular when the label-setting algorithm is used for solving the ESPPRC. To reduce these effects, a modified dominance criteria for the label-setting algorithm is proposed in [25]. Moreover, in order to keep the number of valid inequalities small, a few requirements can be imposed in the separation subproblem. Similarly to [27], we accept only the inequalities which are violated by at least a predefined threshold ε_v. These inequalities are sorted in descending order of their violation and, then, the first K_v inequalities are added to the RMP. A customer is allowed to belong to at most K_s of the subsets that are used to generate the valid inequalities. Moreover, as proposed in Section 3.2, the separation subproblem is called only after the relative gap in the column and cut generation procedure falls below the threshold value ε_c.

4.3. Branching on the VRPTW

Different branching rules have been proposed for branch-and-price methods for solving the VRPTW. In our implementation, we follow the scheme proposed in [27] which is based on the variables of the compact formulation of the VRPTW. Moreover, we use the two-step procedure described in Section 3.3. In the first step, the column and cut generation algorithm is stopped with a loose optimality tolerance $\varepsilon_b > \delta$ and all pricing subproblems are solved by the heuristic label-setting algorithm, *i.e.*, with the simplifications described in Section 4.1. In case it is not possible

to branch, we go to a second phase in which the default optimality tolerance δ is adopted and the exact label-setting algorithm is used when the relative gap falls below δ.

4.4. Rounding heuristic

The use of quick heuristic methods is crucial in the branch-price-and-cut method, specially for identifying integer feasible solutions that can be used to prune nodes. In this context, these heuristics are typically simple procedures that are based on rounding the components of a fractional solution. The use of rounding heuristics within the column generation is usually very helpful [49, 50], as the columns correspond to parts of an integer solution of the original problem. Hence, by putting together a feasible subset of them, an integer solution is likely to be obtained. For instance, in the VRPTW a column corresponds to a feasible route for a vehicle. By selecting a subset of routes (columns) such that each customer is visited exactly once by these routes, we obtain a feasible integer solution. Several heuristics may be used and here we rely on a simple residual rounding heuristic, as it can be easily extended to other integer programming problems. This heuristic can be briefly described as follows. Consider we have a fractional primal solution $\tilde{\lambda}$. The first step is to select up to T non-zero components of $\tilde{\lambda}$ (*e.g.*, $T = 2$). We select the components with the largest values and round them to 1. Then, the columns corresponding to the rounded components are subtracted from b, the right-hand side vector of the RMP. The resulting problem, called residual RMP, is solved again and the process is repeated until an integer solution is obtained or it is detected that no feasible integer solution can be found by the heuristic. It may happen that the residual RMP becomes infeasible and hence, the column generation must be called again until a feasible solution is obtained or the infeasibility is also identified by the column generation procedure. All the columns associated to the rounded components are stored to eventually give rise to an integer feasible solution. Even though a modified RMP is solved inside the heuristic, any column that is generated during its course is also valid for the original RMP.

5. Computational results

In this section, we present the results of computational experiments using the interior point branch-price-and-cut method proposed in this paper. For brevity we will refer to our implementation as IPBPC. We have selected a classical integer programming problem that is widely studied in the branch-and-price literature, the vehicle routing problem with time windows (VRPTW). Of course, having computational experiments for more classes of problems would be very interesting but the difficulty of implementing a branch-price-and-cut method led us to choose one application. Nevertheless, the VRPTW has been used in many previous researches as a benchmark for illustrating and testing new ideas and algorithms. Moreover, the strategies proposed in Section 3 are independent of the problem.

The experiments are run on the benchmarking instances proposed by Solomon [51]. These instances were defined about 30 years ago, but the most difficult ones were solved only in the last few years [52, 53]. They are classified according to the spatial distribution of customers: C instances follow a clustered distribution, R instances follow a random distribution, and RC instances follow a mix of both distributions. As in [27], we focus on the 56 largest instances in the set, all those having 100 customers. These instances are commonly presented in two distinct sets, namely 100-

series and 200-series sets, and those in the second set have wider time windows and larger vehicle capacity, so they are more challenging for the pricing subproblem.

The IPBPC framework has been implemented in the C programming language. It relies on the primal-dual interior point method implemented in the HOPDM code [17, 54] to solve all the linear programming problems. The procedures for solving the pricing and the separation subproblems have also been implemented and they follow the descriptions presented in Sections 4.1 and 4.2. Table 1 shows the default parameter settings adopted in the experiments presented in Section 5.1. For each parameter in the first column, the second column gives the section in which the parameter is defined, and the third column shows its default value. In Section 5.2 we report on additional experiments to verify the impact of some of these parameter choices. The experiments were performed on a Linux PC with an Intel Core i7 2.8 GHz CPU and 8.0 GB of memory.

Parameter	**Section**	**Value**
δ	3.1	10^6
$\varepsilon_{\max}$	3.1	1.0
ε_c	3.2, 4.2	0.1
ε_b	3.3, 4.3	0.001
K_1	4.1	100
K_2	4.1	100
K_d	4.1	0, if $gap > 0.1$ 30, if $0.1 \geq gap > 0.01$ 100, if $0.01 \geq gap$
K_e	4.1	30
K_l	4.1	3 x 10^4, if $gap > 0.1$ 4 x 10^4, if $0.1 \geq gap > 0.01$ 7 x 10^4, if $0.01 \geq gap > 0.001$ 10^5, if $0.001 \geq gap$
K_s	4.2	1
K_v	4.2	3
ε_v	4.2	0.05

Table 1: Parameter choices in the IPBPC implementation for the VRPTW

5.1. Best results and comparison with a simplex-based approach

Tables 2 and 3 show the results for the Solomon's instances in sets 100-series and 200-series, respectively, using the IPBPC with all the features discussed in Sections 3 and 4. In both tables, the first two columns show the name of the instance and the value of the optimal solution obtained by the IPBPC, respectively. In the remaining columns are given the total number of generated columns, the number of generated cuts, the number of nodes in the branch-and-bound tree, the total CPU time spent on the oracle, the total CPU time spent on solving the RMPs, and the total CPU time spent on solving the instance. All the CPU times are given in seconds.

The analysis of these results reveals that the oracle dominates the computational effort, a usual behavior in integer programming problems. Each instance in the 100-series set was solved to optimality in less than 2300 seconds. In the 200-series set, 5 instances could not be solved to optimality due to memory overflow in the pricing subproblem solver called at the last outer iteration of column generation (exact label-setting algorithm). These instances are the same ones that could not be solved in [27] and, hence, it was an expected behavior as the same strategy was used for solving the pricing subproblem in that implementation. Nevertheless, the value of the best integer solution found for each instance is shown in Table 3. For the instances solved to optimality, none of them required more than 17000 seconds.

Instance	Optimal	Columns	Cuts	Nodes	Oracle time	RMP time	Total time
C101	827.3	2165	0	1	0.83	0.28	1.29
C102	827.3	3105	0	1	1.45	0.50	2.20
C103	826.3	3403	0	1	2.95	0.75	3.98
C104	822.9	4280	0	1	7.61	1.29	9.21
C105	827.3	2452	0	1	0.94	0.34	1.48
C106	827.3	2882	0	1	1.21	0.42	1.87
C107	827.3	2734	0	1	1.16	0.40	1.77
C108	827.3	3182	0	1	1.95	0.62	2.84
C109	827.3	3303	0	1	2.20	0.68	3.16
RC101	1619.8	2466	35	1	1.83	1.26	3.23
RC102	1457.4	3923	84	1	7.94	5.58	13.95
RC103	1258.0	5150	130	3	75.97	16.44	94.46
RC104	1132.3	6337	214	9	2230.29	52.02	2292.06
RC105	1513.7	3060	32	1	2.71	1.52	4.40
RC106	1372.7	9896	539	61	634.69	134.59	855.87
RC107	1207.8	3877	60	1	20.60	4.37	25.30
RC108	1114.2	5019	97	1	170.36	11.96	182.71
R101	1637.7	1708	4	3	0.94	0.42	1.69
R102	1466.6	2372	0	1	1.26	0.35	1.74
R103	1208.7	3337	18	1	4.10	1.74	6.18
R104	971.5	5801	184	5	364.24	33.93	411.50
R105	1355.3	2820	43	3	3.02	2.33	6.24
R106	1234.6	3722	52	3	7.77	4.10	13.17
R107	1064.6	4671	127	3	61.03	15.06	78.09
R108	932.1	5548	165	1	391.37	24.66	416.64
R109	1146.9	7010	339	35	234.72	72.51	348.13
R110	1068.0	4506	121	3	44.38	14.46	62.04
R111	1048.7	18186	672	107	1336.66	206.39	1701.96
R112	948.6	7857	415	17	1403.98	127.79	1573.75
Total		134772	3331	269	7018.14	736.73	8120.90

Table 2: IPBPC results for the 100-series Solomon's instances.

Instance	Optimal	Columns	Cuts	Nodes	Oracle time	RMP time	Total time
C201	589.1	2718	0	1	5.07	1.52	7.02
C202	589.1	5766	0	1	11.25	5.66	17.37
C203	588.7	7552	0	1	29.02	10.96	40.64
C204	588.1	11451	0	1	710.92	29.63	741.33
C205	586.4	3344	0	1	5.05	2.01	7.41
C206	586.0	5346	0	1	8.30	4.48	13.22
C207	585.8	6230	0	1	13.76	6.59	20.90
C208	585.8	4884	0	1	12.17	4.91	17.49
RC201	1261.8	9009	17	3	23.58	14.75	42.27
RC202	1092.3	9648	9	1	34.43	17.34	53.15
RC203	923.7	12238	6	2	305.02	34.56	341.50
RC204	783.5*						
RC205	1154.0	8250	9	1	24.68	11.37	36.69
RC206	1051.1	9430	9	1	61.51	22.00	84.27
RC207	962.9	14622	116	3	2487.92	190.59	2691.81
RC208	776.1*						
R201	1143.2	8466	20	1	21.05	14.58	36.24
R202	1029.6	12461	62	9	89.49	51.79	147.21
R203	870.8	14117	18	1	295.70	44.32	342.07
R204	731.3*						
R205	949.8	17262	112	13	738.09	193.86	968.18
R206	875.9	18307	81	3	2042.07	113.64	2166.28
R207	794.0	15090	9	1	5434.13	65.78	5501.94
R208	700.2*						
R209	854.8	15609	97	5	1073.17	137.10	1220.33
R210	900.5	16395	126	9	16454.83	280.83	16783.85
R211	746.7*						
Total		228195	691	61	29881.19	1258.28	31281.16

* The instance was not solved to optimality due to memory overflow. This value corresponds to the best incumbent solution that was found by the IPBPC.

Table 3: IPBPC results for the 200-series Solomon's instances.

It is useful to compare the performance of the IPBPC with a similar branch-price-and-cut method for the VRPTW that uses the simplex method to solve linear programming problems. Such comparison allows us to verify if the use of an interior point algorithm within a branch-price-and-cut method is indeed worthwhile. Hence, in Tables 4 and 5 we show the results of the simplex-based branch-price-and-cut method proposed in [27], and a comparison with the results shown in Tables 2 and 3. The simplex-based implementation uses the same type of valid inequalities as presented in Section 4.2 and the pricing subproblem is solved using a procedure very similar to that described in Section 4.1. However, the results in [27] were run on a different computer, namely a Linux PC with a Dual Core AMD Opteron 2.6 GHz CPU, and hence the conclusions about CPU time should be taken cautiously. The first few columns in Tables 4 and 5 give the number of cuts, number of nodes and total CPU times, as presented in [27]. The remaining columns give the ratio between the values in the first columns and the corresponding values in Tables 2 and 3. Only the instances that were successfully solved by both approaches are presented in the tables.

For each 100-series instance, the IPBPC generated not only a smaller (or equal) number of valid inequalities, but also a smaller (or equal) number of nodes when compared with the simplex-based BPC. In total, the simplex-based BPC required around 68% more valid inequalities and 31% more nodes. The IPBPC was about 5 times faster than the other approach, but the reader should be warned that the two approaches were run on different computers (we did not have access to the proprietary code described in [27]). Concerning the results for the 200-series instances, the total number of nodes in the IPBPC was a bit larger than the number of nodes in the simplex-based approach in this case. On the other hand, the number of valid inequalities is still smaller for every instance and there is an even larger difference when compared to the number of valid inequalities generated by the simplex-based BPC. For the instance RC203, for example, the simplex-based method generated almost 8 times more valid inequalities than the IPBPC. Large differences are observed for other instances as well, such as RC202 and RC206, which reveals an important advantage of the IPBPC. As discussed in Section 3.2, keeping the number of valid inequalities small is crucial for the efficiency of the subproblem solver, as the SR inequalities affect the structure of the subproblem. Considering the total CPU time, the IPBPC outperforms the simplex-based BPC. These results indicate that the use of the primal-dual interior point algorithm within the branch-price-and-cut method offers a significant advantage over the simplex-based approach.

5.2. Impact of changes in the core components

To verify the importance of the modifications proposed in this paper for the core components of the IPBPC, we have run some additional computational experiments with different choices of parameters. With these experiments we do not aim to have an exhaustive testing of all possible parameter choices, but rather we want to investigate the impact of changing some key parameters of the method to get a better understanding of the proposed strategy.

In the first experiment, we analyze how the early branching strategy described in Sections 3.3 and 4.3 affects the performance of the IPBPC. In Fig. 1 we plot the results of solving all the 100-series instances with different choices of the parameter ε_b, the threshold tolerance for finishing the preprocessing step in the two-step branching approach. Four different values of ε_b were tested: 10^{-3}, 10^{-4}, 10^{-5} and 10^{-6}. Three plots are given in Fig. 1 and they show (a) the total number of valid inequalities, (b) the total number of nodes in the search tree, and (c) the total CPU time, for each choice of ε_b. From (a), we see that the values 10^{-3} and 10^{-4} resulted in the smallest numbers of valid inequalities.

	Simplex-based BPC [27]			Ratio		
	Cuts	**Nodes**	**Total time**	**Cuts**	**Nodes**	**Total time**
C101	0	1	2	1.00	1.00	1.56
C102	0	1	8	1.00	1.00	3.63
C103	0	1	28	1.00	1.00	7.04
C104	0	1	86	1.00	1.00	9.34
C105	0	1	3	1.00	1.00	2.03
C106	0	1	4	1.00	1.00	2.13
C107	0	1	4	1.00	1.00	2.27
C108	0	1	7	1.00	1.00	2.47
C109	0	1	16	1.00	1.00	5.06
RC101	87	1	19	2.49	1.00	5.88
RC102	193	3	120	2.30	3.00	8.60
RC103	262	5	541	2.02	1.67	5.73
RC104	437	21	11773	2.04	2.33	5.14
RC105	79	1	33	2.47	1.00	7.49
RC106	755	71	3916	1.40	1.16	4.58
RC107	158	1	161	2.63	1.00	6.36
RC108	228	1	635	2.35	1.00	3.48
R101	19	15	8	4.75	5.00	4.74
R102	0	1	3	1.00	1.00	1.72
R103	53	1	20	2.94	1.00	3.24
R104	391	11	3103	2.13	2.20	7.54
R105	144	3	36	3.35	1.00	5.77
R106	144	3	87	2.77	1.00	6.61
R107	227	4	416	1.79	1.33	5.33
R108	296	1	891	1.79	1.00	2.14
R109	588	65	1127	1.73	1.86	3.24
R110	219	5	426	1.81	1.67	6.87
R111	736	111	5738	1.10	1.04	3.37
R112	574	19	16073	1.38	1.12	10.21
Total	5590	352	45284	1.68	1.31	5.58

Table 4: Comparison to a simplex-based BPC method (100-series Solomon's instances).

	Simplex-based BPC [27]			Ratio		
	Cuts	**Nodes**	**Total time**	**Cuts**	**Nodes**	**Total time**
C201	0	1	9	1.00	1.00	1.28
C202	0	1	49	1.00	1.00	2.82
C203	0	1	122	1.00	1.00	3.00
C204	0	1	16416	1.00	1.00	22.14
C205	0	1	15	1.00	1.00	2.02
C206	0	1	24	1.00	1.00	1.82
C207	0	1	84	1.00	1.00	4.02
C208	0	1	26	1.00	1.00	1.49
RC201	55	3	92	3.24	1.00	2.18
RC202	39	1	89	4.33	1.00	1.67
RC203	47	1	324	7.83	1.00	0.95
RC205	32	1	111	3.56	1.00	3.03
RC206	73	1	344	8.11	1.00	4.08
RC207	210	5	91405	1.81	1.67	33.96
R201	52	1	78	2.60	1.00	2.15
R202	152	17	1663	2.45	1.89	11.30
R203	78	1	641	4.33	1.00	1.87
R205	345	9	6904	3.08	0.69	7.13
R206	171	1	60608	2.11	0.33	27.98
R207	24	1	11228	2.67	1.00	2.04
R209	248	3	22514	2.56	0.60	18.45
R210	266	5	400904	2.11	0.56	23.89
Total	1792	58	613650	2.59	0.97	19.62

Table 5: Comparison to a simplex-based BPC method (200-series Solomon's instances).

From (b) and (c), we deduce that $\varepsilon_b = 10^{-3}$ resulted in the smallest number of nodes and the best total CPU time. The use of $\varepsilon_b = 10^{-6}$ in the branching strategy causes a loss of efficiency measured in nearly doubling the CPU time when compared with using $\varepsilon_b = 10^{-3}$. Hence, we conclude that the early branching contributed to the overall efficiency of the IPBPC. Notice that a reduction in the CPU time was expected for larger values of ε_b, as the column and cut generation is stopped earlier and, hence, fewer calls to the oracle are made. However, the reduction in the number of nodes is an interesting result and it suggests that, indeed, interior point methods have the advantage of quickly approaching the optimal solution. As a consequence of this feature, the provided suboptimal solutions are accurate enough to be safely used in the branching procedure.

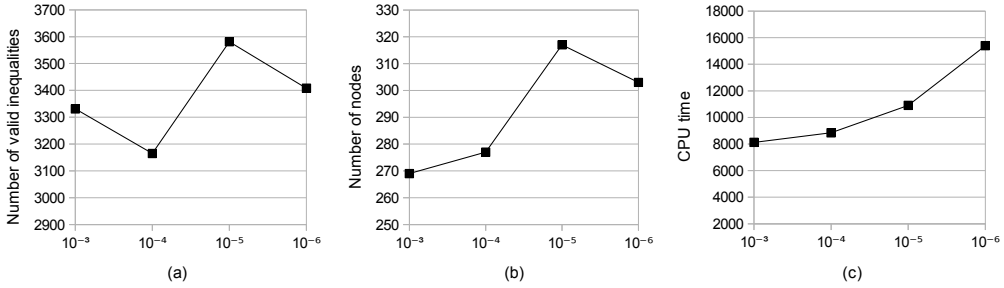

Figure 1: Impact of changing the early branching threshold ε_b.

The second experiment involves the impact of the parameter ε_c which is used in the oracle of the column and cut generation procedure described in Section 3.2. Recall that this parameter has the following purpose: the separation subproblem is called by the oracle only after the relative gap falls bellow ε_c. Three different values of ε_c were tested in this experiment, namely 0.1, 0.01 and 0.001. In addition, recall that after calling the separation subproblem, up to K_v valid inequalities are added to the RMP. The default value $K_v = 3$ is used in the IPBPC, but for a small ε_c it may be better to allow a larger number of valid inequalities to be added to the RMP. Hence, we have run tests with $K_v = 10$ as well. Fig. 2 summarizes the results that were obtained in this experiment. It has three plots with the same meaning as those given in Fig. 1. Each plot shows the results for different choices of ε_c; the continuous line corresponds to $K_v = 3$ while the dashed line corresponds to $K_v = 10$. The analysis of these results suggests that the best approach corresponds to calling the separation subproblem as soon as the relative gap in the column and cut generation procedure falls below 0.1. For this scenario, $K_v = 3$ seems to be a more appropriate choice than $K_v = 10$, as the latter resulted in a larger number of generated valid inequalities and larger number of nodes. In addition, the CPU time was considerably increased. The results for $K_v = 10$ were improved when ε_c was reduced, but still they were inferior to the overall performance obtained with $K_v = 3$.

6. Conclusions

In this paper, we have addressed the use of the primal-dual interior point algorithm within the branch-price-and-cut method. For each core component of the method, namely the column generation procedure, the separation of valid inequalities (cuts) and the branching procedure, we have presented how to exploit certain advantages that are provided

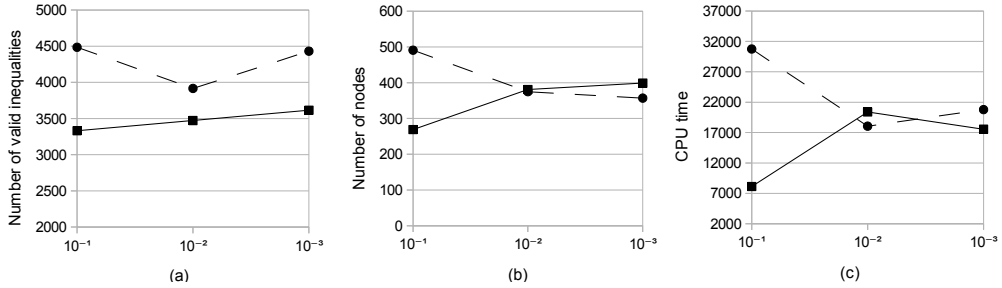

Figure 2: Impact of changing the separation subproblem threshold ε_c.

by the interior point algorithm. Two of them are particularly attractive in this context: (*i*) the ability to work with well-centered solutions stabilizes the column and cut generation procedure; (*ii*) the use of early termination improves the overall efficiency of the approach. To verify the behavior of the proposed interior point branch-price-and-cut method, we have run computational experiments on well-known instances of the vehicle routing problem with time windows, a classical integer programming problem. The results provide evidence that the proposed method is efficient and outperforms a similar branch-price-and-cut method which uses the simplex algorithm to solve linear programming problems. We have observed considerable reductions in the number of nodes in the search tree, in the number of generated valid inequalities and in the CPU time to solve the instances.

The next steps regarding this investigation will involve solving different integer programming problems by the interior point branch-price-and-cut method proposed in this paper. The focus will be on problems with a large-scale master problem formulation, so that a larger percentage of CPU time is spent on solving the RMPs. Also, we intend to improve our ESPPRC implementation by following the main ideas that were recently proposed in [52, 53] in order to solve the Solomon's instances that could not be solved by the current implementation.

Acknowledgements

The authors would like to thank the anonymous referee for his/her valuable contributions to this paper.

References

[1] J. Desrosiers, M. E. Lübbecke, Branch-Price-and-Cut Algorithms, Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc., 2011.

[2] G. Desaulniers, J. Desrosiers, S. Spoorendonk, Cutting planes for branch-and-price algorithms, Networks 58 (4) (2011) 301–310.

[3] F. Vanderbeck, L. A. Wolsey, Reformulation and decomposition of integer programs, in: M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, L. A. Wolsey (Eds.), 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, 2010, pp. 431–502.

[4] O. Briant, C. Lemarechal, P. Meurdesoif, S. Michel, N. Perrot, F. Vanderbeck, Comparison of bundle and classical column generation, Mathematical Programming 113 (2) (2008) 299–344.

[5] H. M. Ben Amor, J. Desrosiers, A. Frangioni, On the choice of explicit stabilizing terms in column generation, Discrete Applied Mathematics 157 (6) (2009) 1167 – 1184.

[6] J. L. Goffin, J. P. Vial, Convex nondifferentiable optimization: a survey focussed on the analytic center cutting plane method, Optimization Methods and Software 17 (5) (2002) 805–867.

[7] A. Frangioni, Generalized bundle methods, SIAM Journal on Optimization 13 (2002) 117–156.

[8] J. Gondzio, P. González-Brevis, P. Munari, New developments in the primal-dual column generation technique, European Journal of Operational Research 224 (1) (2013) 41–51.

[9] P. Benchimol, G. Desaulniers, J. Desrosiers, Stabilized dynamic constraint aggregation for solving set partitioning problems, European Journal of Operational Research 223 (2) (2012) 360 – 371.

[10] J. E. Mitchell, M. J. Todd, Solving combinatorial optimization problems using Karmarkar's algorithm, Mathematical Programming 56 (1992) 245–284.

[11] J. Mitchell, B. Borchers, Solving real-world linear ordering problems using a primal-dual interior point cutting plane method, Annals of Operations Research 62 (1996) 253–276.

[12] J. E. Mitchell, Computational experience with an interior point cutting plane algorithm, SIAM Journal of Optimization 10 (4) (2000) 1212–1227.

[13] S. Elhedhli, J.-L. Goffin, The integration of an interior-point cutting plane method within a branch-and-price algorithm, Mathematical Programming 100 (2004) 267–294.

[14] J. Gondzio, Warm start of the primal-dual method applied in the cutting plane scheme, Mathematical Programming 83 (1) (1998) 125–143.

[15] H. Y. Benson, D. F. Shanno, An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming, Computational Optimization and Applications 38 (3) (2007) 371–399.

[16] A. Engau, M. F. Anjos, A. Vannelli, On interior-point warmstarts for linear and combinatorial optimization, SIAM Journal on Optimization 20 (4) (2010) 1828–1861.

[17] J. Gondzio, Interior point methods 25 years later, European Journal of Operational Research 218 (3) (2012) 587–601.

[18] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984) 373–395.

[19] S. J. Wright, Primal-dual interior-point methods, SIAM, Philadelphia, 1997.

[20] B. Borchers, J. E. Mitchell, Using an interior point method in a branch and bound algorithm for integer programming, Tech. Rep. 195, Mathematical Sciences, Rensselaer Polytechnic Institute (July 1992).

[21] O. du Merle, P. Hansen, B. Jaumard, N. Mladenovic, An interior point algorithm for minimum sum-of-squares clustering, SIAM J. Sci. Comput. 21 (4) (1999) 1485–1505.

[22] D. Aloise, P. Hansen, L. Liberti, An improved column generation algorithm for minimum sum-of-squares clustering, Mathematical Programming 131 (2012) 195–220.

[23] H. Ben Amor, J. Desrosiers, A proximal trust-region algorithm for column generation stabilization, Computers and Operations Research 33 (4) (2006) 910 – 927.

[24] S. Spoorendonk, Cut and column generation, Ph.D. thesis, Technical University of Denmark, Department of Management Engineering (2008).

[25] M. Jepsen, B. Petersen, S. Spoorendonk, D. Pisinger, Subset-row inequalities applied to the vehicle-routing problem with time windows, Operations Research 56 (2) (2008) 497–511.

[26] D. Villeneuve, J. Desrosiers, M. Lübbecke, F. Soumis, On compact formulations for integer programs solved by column generation, Annals of Operations Research 139 (2005) 375–388.

[27] G. Desaulniers, F. Lessard, A. Hadjar, Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows, Transportation Science 42 (3) (2008) 387–404.

[28] E. A. Yildirim, S. J. Wright, Warm-start strategies in interior-point methods for linear programming, SIAM Journal on Optimization 12 (3) (2002) 782–810.

[29] J. Gondzio, A. Grothey, Reoptimization with the primal-dual interior point method, SIAM Journal on Optimization 13 (3) (2003) 842–864.

[30] E. John, A. Yildirim, Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension, Computational Optimization and Applications 41 (2008) 151–183.

[31] J. Gondzio, A. Grothey, A new unblocking technique to warmstart interior point methods based on sensitivity analysis, SIAM Journal on Optimization 19 (3) (2008) 1184–1210.

[32] J. Gondzio, Multiple centrality corrections in a primal-dual method for linear programming, Computational Optimization and Applications 6 (2) (1996) 137–156.

[33] M. Colombo, J. Gondzio, Further development of multiple centrality correctors for interior point methods, Computational Optimization and Applications 41 (2008) 277–305.

[34] O. Braysy, M. Gendreau, Vehicle routing problem with time windows, Part I: Route construction and local search algorithms, Transportation Science 39 (1) (2005) 104–118.

[35] G. Desaulniers, J. Desrosiers, S. Spoorendonk, The vehicle routing problem with time windows: State-of-the-art exact solution methods, in: J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, J. C. Smith (Eds.), Wiley Encyclopedia of Operations Research and Management Science, Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc., 2010.

[36] V. Pureza, R. Morabito, M. Reimann, Vehicle routing with multiple deliverymen: Modeling and heuristic approaches for the VRPTW, European Journal of Operational Research 218 (3) (2012) 636 – 647.

[37] R. Macedo, C. Alves, J. Valerio de Carvalho, F. Clautiaux, S. Hanafi, Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model, European Journal of Operational Research 214 (3) (2011) 536–545.

[38] B. Kallehauge, J. Larsen, O. B. Madsen, M. M. Solomon, Vehicle routing problem with time windows, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer US, 2005, pp. 67–98.

[39] D. Feillet, A tutorial on column generation and branch-and-price for vehicle routing problems, 4OR: A Quarterly Journal of Operations Research 8 (2010) 407–424.

[40] M. Desrochers, An algorithm for the shortest path problem with resource constraints, Tech. rep., Technical Report G-88-27, GERAD (1988).

[41] J. E. Beasley, N. Christofides, An algorithm for the resource constrained shortest path problem, Networks 19 (4) (1989) 379–394.

[42] D. Feillet, P. Dejax, M. Gendreau, C. Gueguen, An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle-routing problems, Networks 44 (2004) 216–229.

[43] G. Righini, M. Salani, New dynamic programming algorithms for the resource constrained elementary shortest path problem, Networks 51 (3) (2008) 155–170.

[44] A. Chabrier, Vehicle routing problem with elementary shortest path based column generation, Computers and Operations Research 33 (10) (2006) 2972 – 2990.

[45] G. Clarke, J. W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Operations Research 12 (4) (1964) 568–581.

[46] M. Dror, Note on the complexity of the shortest-path models for column generation in VRPTW, Operations Research 42 (5) (1994) 977–978.

[47] N. Kohl, J. Desrosiers, O. Madsen, M. Solomon, F. Soumis, 2-path cuts for the vehicle routing problem with time windows, Transportation Science 33 (1) (1999) 101–116.

[48] W. Cook, J. L. Rich, A parallel cutting plane algorithm for the vehicle routing problem with time windows, Tech. Rep. TR99-04, Computational and Applied Mathematics, Rice University (May 1999).

[49] F. Vanderbeck, Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem, Oper. Res. 48 (2000) 915–926.

[50] Z. Degraeve, M. Peeters, Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results, INFORMS Journal on Computing 15 (1) (2003) 58–81.

[51] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35 (2) (1987) 254–265.

[52] R. Baldacci, A. Mingozzi, R. Roberti, Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints, European Journal of Operational Research 218 (1) (2012) 1–6.

[53] S. Ropke, Branching decisions in branch-and-price-and-cut algorithms for vehicle routing problems, in: International Workshop on Column Generation, Bromont, Canada, June 10-13, 2012.

[54] J. Gondzio, HOPDM (version 2.12) - a fast LP solver based on a primal-dual interior point method, European Journal of Operational Research 85 (1) (1995) 221–225.